BY FRED EADY

BRINGING A USB-TO-UART PROTOCOL CONVERTER TO LIFE

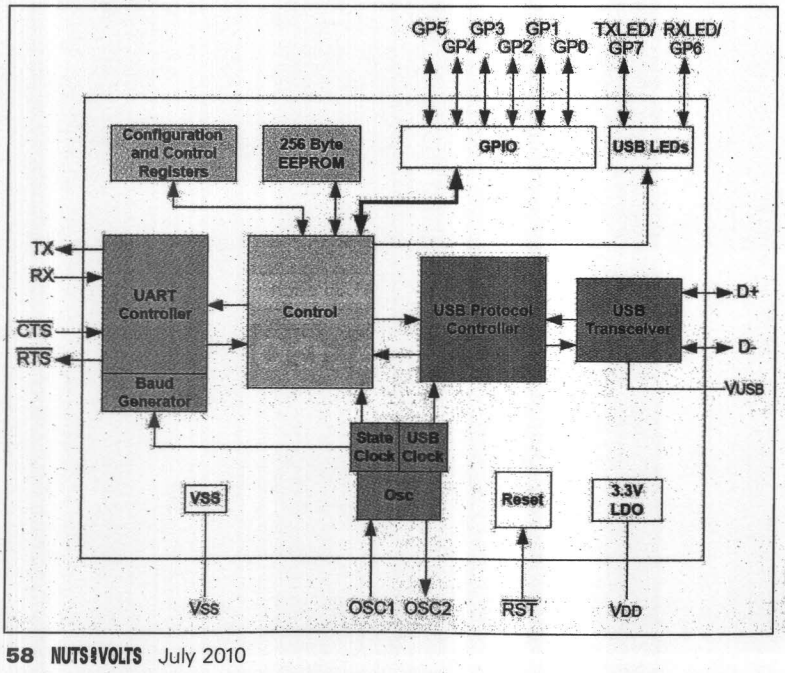
Did you know that Microchip offers a USB 2.0 to UART protocol converter? It's called the MCP2200. The MCP2200 is not totally limited to performing USB-to-serial conversion duty. Outfitted in 7.5 mm (.300 inches) SOIC packaging, the MCP2200 resembles a PIC in form and function by providing an octet of GPIO (General-Purpose Input/Output) pins and EEPROM. In this edition of Design Cycle, you and I are going to scratch-design a plug-in USB converter module based on the MCP2200. After we bring the MCP2200 hardware to life, we'll configure the MCP2200 and put it to work in front of the UART of a USB-challenged microcontroller.

MCP2200 ORIENTATION

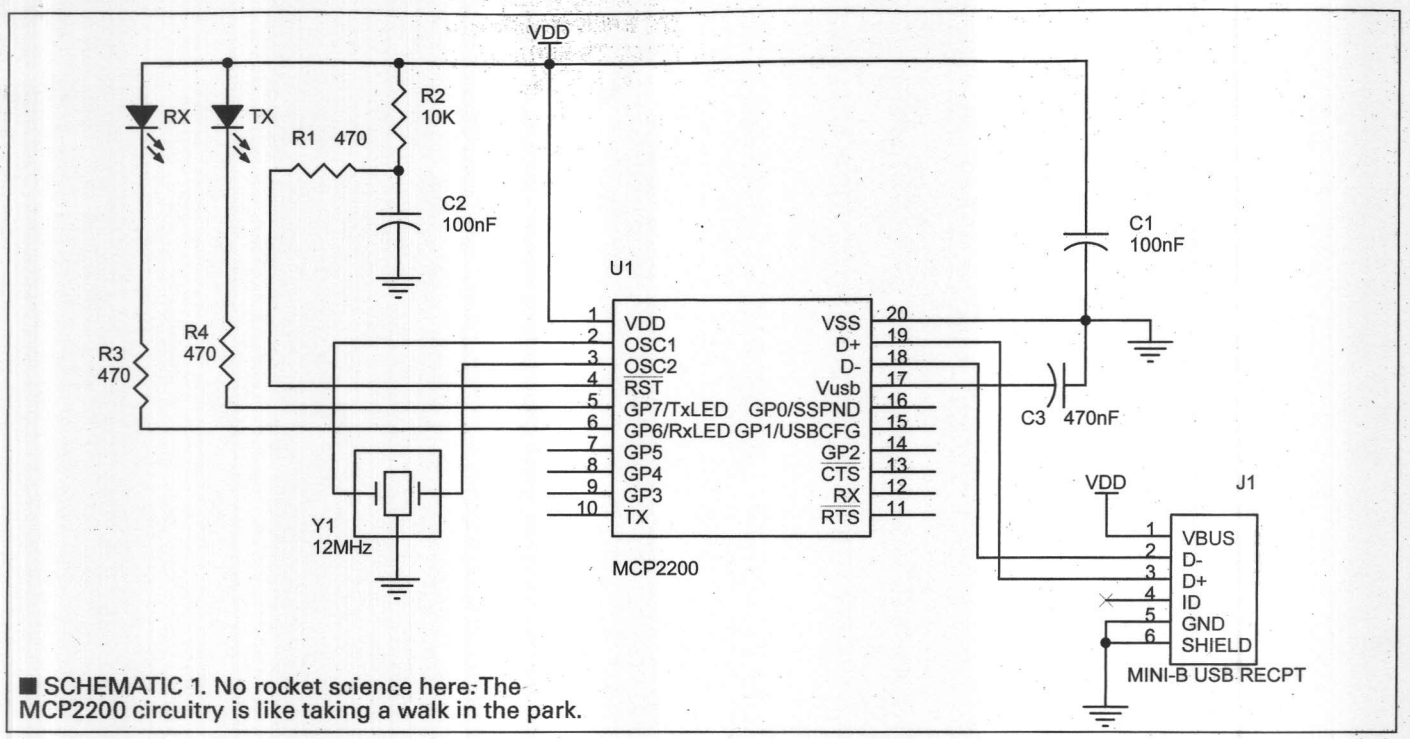
In addition to SOIC, the MCP2200 can be had in QFN and SSOP packages. If you're a hobbyist, chances are you don't have the necessary soldering tools to lay down tiny leadless QFN packages. The chances are also slim as to a hobbyist's toolability to accurately place and hand-solder SSOP parts. So, I've chosen to develop our MCP2200 design around the larger and easier to handle 20-pin SOIC package which can be easily placed and hand-soldered.

The MCP2200 is designed to support full speed 12 Mbps USB applications. If you go back a few issues (*Nuts & Volts* June '09 to be exact), you'll see that we've been actively pursuing USB trickery as it pertains to RS-232 for some time now. In the June '09 Design Cycle, we used a PIC low pin count USB development kit to form the basis of a USB CDC (Communications Device Class) device. The MCP2200 is also a USB protocol composite CDC device. One logical portion of the MCP2200 is a HID-class device and the other part is a CDC device. Thus, we will need to supply a driver for the CDC device. Recall that HID-class device drivers are part of the operating system. Not to worry. We won't be writing any MCP2200 CDC driver code as Microchip provides the necessary drivers.

The MCP2200 differs from the low pin count PIC18F14K50 we used in June '09 in that the MCP2200 is a dedicated USB device and cannot be programmed to run user-specific application firmware. For instance, we had to program the PIC18F14K50 to activate an I/O pin when it successfully enumerated to the CONFIGURED USB state. In the PIC18F14K50 design, we used the activated I/O pin to enable a 5.0 volt voltage regulator which provided power for auxiliary five volt electronic devices. We can perform the same "activate I/O pin when configured" functionality by simply utilizing the MCP2200's USBCFG status pin.



■ FIGURE 1. It looks like a PIC, kinda acts like a PIC, but it ain't officially a PIC. At least the datasheet says it ain't one.



■ SCHEMATIC 1. No rocket science here: The MCP2200 circuitry is like taking a walk in the park.

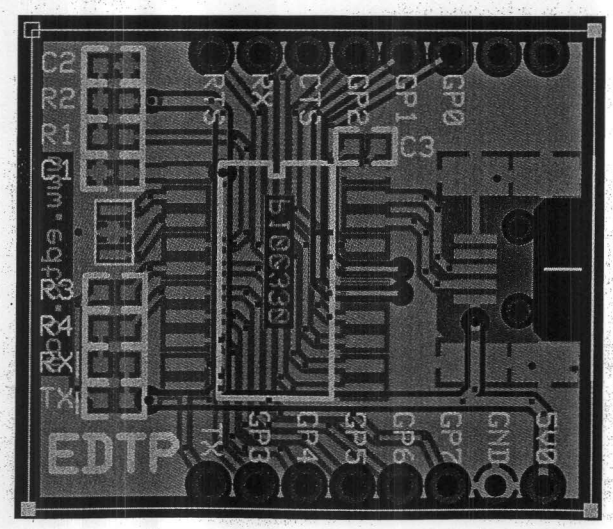
A high-level block diagram of the MCP2200 is depicted in **Figure 1**. The MCP2200 sure looks like a "specialized" microcontroller. However, I can't yet prove that it is. So, let's take a walk around its pins.

The MCP2200 can operate with voltages between 3.0 volts and 5.5 volts. In that most low power embedded applications that will employ the MCP2200 draw their power directly from the USB portal, the MCP2200 will find itself powered by the USB portal's 5.0 volt V_{BUS} line in most instances. This mode of operation is termed Bus Power Only mode in the MCP2200 datasheet. The MCP2200 can also be configured in self-powered mode.

I have this microcontroller feel about the MCP2200. One thing that really makes me think PIC is the MCP2200's need for an external 12 MHz quartz crystal or ceramic resonator. However, my "It's really a PIC" theory could be debunked by getting up on my donkey and declaring the clock signal is needed for the USB interface and/or the MCP2200's internal controller. Judging from the oscillator block in **Figure 1**, I'm willing to bet that the 12 MHz clock signal is processed through a 4x PLL to synthesize a 48 MHz USB clock. I'm also betting the farm on the control block really being a microcontroller block as it supports the configuration registers (SFRs in the PIC world), EEPROM, and GPIO. I'll get down off my donkey and continue our ceramic oscillator tale. The MCP2200 datasheet calls out a muRata CSTCE12M0G15L; the MCP2200 Demo Board User's Guide specifies a muRata CSTCE12M0G15L99-R0. I couldn't find either of those ceramic oscillators that I didn't have to buy 3,000 of. So, I finally came up with a muRata CSTCE12M0G55-R0 (Digi-Key part number 490-1197-1-ND) that we will use in our design.

Here we go again. Take a look at **Schematic 1**. Doesn't that arrangement of R1, R2, and C2 look like it could hang from a PIC's MCLR pin? The datasheet says we can sack C2. However, it's a good idea to have it in our design as it doesn't interfere with the operation of the MCP2200's internal POR circuitry. If you're absolutely sure your MCP2200 design will be driven by a host microcontroller with an available I/O line, you can dispense with the R2/C2 combination and drive the MCP2200's active-low RST pin directly from a microcontroller I/O pin.

If you've ever read a Design Cycle column, you know that I LOVE to blink LEDs. The MCP2200 I/O pins that interface to the LEDs in our design are part of its GPIO module. Like a microcontroller, the MCP2200's GPIO module is a standard eight-bit port. (Hmmmm ... looks like a microcontroller and is beginning to smell of one too.) Some of the GPIO module's I/O pins have alternate functions; this includes



■ SCREENSHOT 1. We could shrink this design considerably by replacing the SOIC with a QFN and reducing the SMT component size from 0603 to 0402. However, I think it's small enough right now, thank you.

the GP7/TxLED and GP6/ RxLED I/O pins. The blink pulse width of the TX and RX LEDs is configurable. In fact, you can configure the TX and RX LEDs to toggle on each message which allows your application to count incoming and outgoing messages by simply counting the LED toggles.

While we're on the subject of alternate GPIO pins, GP0 doubles as the USB suspend status pin. When the USB link goes into suspend mode, the MCP2200's USB suspend status pin can be used as a signal to put the entire device into low power mode. We briefly touched on the use of the MCP2200's USBCFG pin which is the alter ego of GPIO pin GP1. GPIO pins GP2 through GP5 are single-minded GPIO pins with no ulterior motives.

Years ago, I wrote an interrupt-driven UART receive/transmit routine that buffered incoming and outgoing bytes. I still use that code today. The idea was to not miss an incoming byte due to a more important process that happened to be taking up the CPU's time at that moment. On the transmit side, the CPU could post a byte to be transmitted into the buffer and return to business as usual without having to immediately worry about walking the byte through the entire transmission process. In that we can't code the MCP2200 and RS-232 I/O data buffering is important in high throughput applications, the MCP2200 engineers built a 128-byte buffer into its innards. The MCP2200's UART buffer is equally divided into 64 bytes for transmit and 64 bytes for receive operations. With the assistance of the UART buffer, the MCP2200 can support baud rates between 300 and 1 Mbps on its TX and RX pins.

In the golden days of the BBS masters, one had to be familiar with modems and modem control lines. Back then, it was a must to know that the DTR (Data Terminal Ready) signal had to trigger a DSR (Data Set Ready) signal from the modem before anything else could happen. If you were the "master" of a popular BBS, the RI (Ring Indicate) signal was a constant modem companion. RTS (Request To Send) and CTS (Clear To Send) are still in use today as hardware flow control signals for embedded devices. With that, the MCP2200's I/O subsystem and hardware flow control logic are equipped to handle situations where RTS/CTS hardware flow control is employed.

In the case of RTS/CTS flow control MCP2200 style, the MCU host's active-low CTS input is connected to the MCP2200's active-low RTS output. This active-low RTS signal is used by the MCP2200 to signal the host device that it can receive data. If the host gets chatty, the MCP2200 will raise the RTS signal one byte short of filling its buffer (63 bytes).

The MCP2200 CTS input is connected to the host MCU's active-low RTS output. When the MCP2200 sees its CTS input go logically low, it will send data. Once the send operation is initiated, raising the CTS signal will not abort the transfer that is in progress.

If the MCP2200 flow control mechanism is disabled, the buffer pointer does not increment. Without the guidance of the buffer pointer, the buffer can be overrun. The result of an MCP2200 buffer overrun is that the new data will overwrite the last position of the buffer.

Okay. Now that you have a rough idea of the MCP2200's capabilities, let's build that MCP2200 dongle.

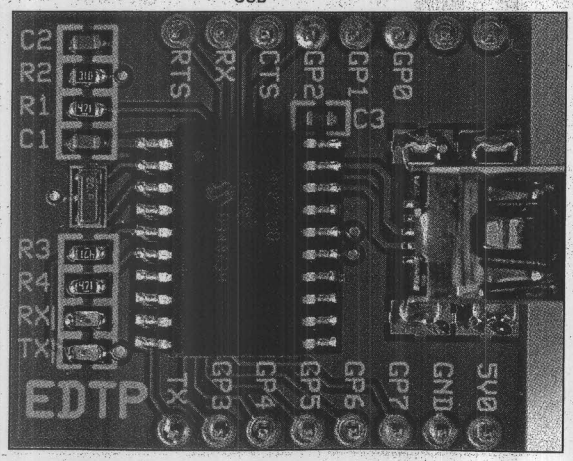
AN MCP2200 HARDWARE DESIGN

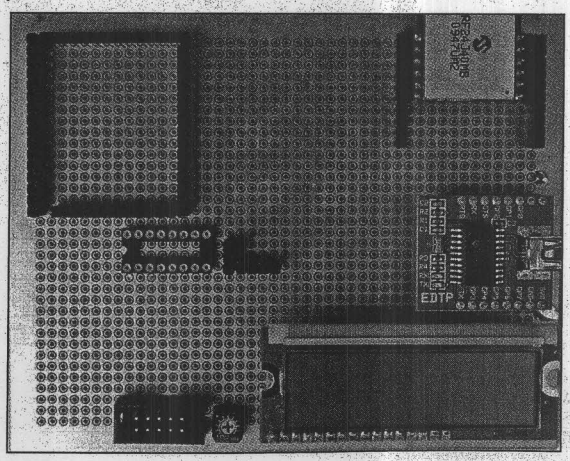
The MCP2200 USB-to-UART converter design outlined in **Schematic 1** could easily be adapted to a breadboard. However, the SMT components lend themselves to being mounted on a specialized printed circuit board (PCB) like the one you see on the drawing board in **Screenshot 1**.

There's not much I can tell you about assembling the MCP2200 USB-to-UART converter that you don't already know. As with any electronic project, keep your mind glued to the details to avoid releasing the magic smoke. There are no component polarity gotchas in this design as all of the capacitors are nonpolarized ceramic types. You do need to be careful when mounting the MCP2200 as you must pay attention to the correct orientation of pin 1 of the MCP2200. If your LEDs fail to blink, check to make sure that the cathodes of the LEDs are on the bar sides of the LED pads. The LED cathode bars are just to the right of the RX and TX silkscreen legends.

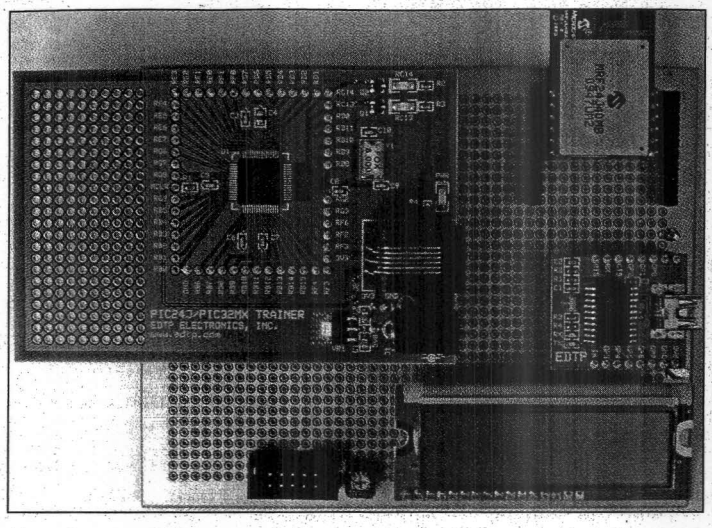
The MCP2200 USB-to-UART converter headers are on 0.1 inch centers and will mate with any standard breadboard or solderless breadboard similarly pitched. A fully assembled MCP2200 USB-to-UART converter with header pins is

■ PHOTO 1. Here's a fully assembled MCP2200 USB-to-UART converter reporting for duty. All of the GPIO pins and the TX and RX pins are terminated at a header pad. There's even a five volt header pad that makes the five volts supplied via V_{USB} available to external circuitry.





■ PHOTO 2. If you've been keeping up with Design Cycle and *SERVO Magazine*, you're already familiar with this piece of golden perfboard. I've removed the SP3232 RS-232 converter IC and replaced it with our MCP2200 USB-to-UART converter design.



■ PHOTO 3. Here's the whole shebang. A MCP2200 USB-to-UART converter-equipped PIC24FJ/PIC32MX Trainer in command of an MRF24J40MB 802.15.4 transceiver and an LCD.

shown in **Photo 1**. The idea behind the MCP2200 is to replace legacy RS-232 installations. The board you see in **Photo 2** used to have an SP3232 RS-232 converter IC in that empty socket. I installed the new MCP2200 USB-to-UART converter with only four connections. The converter's TX pin connects to the PIC24FJ128GA006's UART RX pin, and the PIC24FJ128GA006's UART RX pin crosses over to the converter's TX pin. The MCP2200 converter's five volt power and ground connections complete the connection quartet. The entire design pictured in **Photo 3** is graphically displayed in **Schematic 2**. Everything hardware is in place. So, let's plug a USB cable into the MCP2200 USB-to-UART Converter and see what happens.

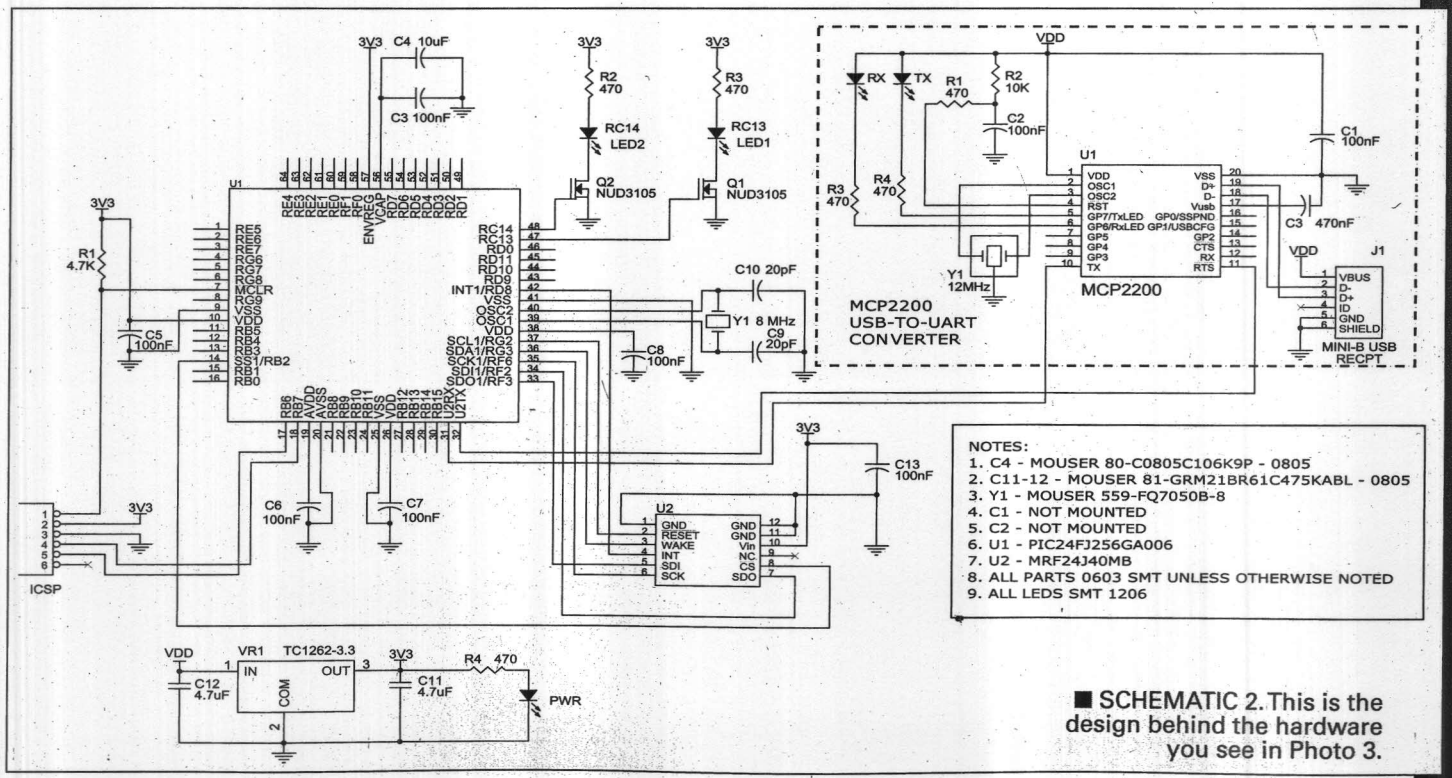
THE MCP2200 CONFIGURATION UTILITY

After attaching a USB cable to the MCP2200 USB-to-UART converter, I fired up the Configuration Utility and captured the main configuration window in **Screenshot 2**. The default VID (Vendor ID) and PID (Product ID) belong to Microchip. Obviously, if you owned your own VID and PID, you would enter them here.

If you were to expand the Baud Rate menu, you would find the highest baud rate value is 921600. The list of baud rates between 9600 and 921600 are the standard fare as far as baud rates go. For the purposes of our discussion, 9600 bps is plenty fast.

Once again the MCP2200 is starting to really look and feel like a PIC. The I/O config field is an eight-bit mask. A logical 0 says that the bit position is an output while a logical 1 sets the associated bit location as an input. Funny, that's exactly how PIC TRIS registers work. The Output Default window sets the logical output of any MCP2200 GPIO pin defined as an output pin in the I/O config field.

The rest of the Configuration Utility is intuitively obvious to the most casual observer. The alternate GPIO pin selects live in the upper right corner. I pulled the trigger on the displayed configuration in **Screenshot 3**. Before we leave the Configuration Utility, note that we can invert the UART signal polarity. The UPOL selection saves hardware by eliminating the need to place inverters in the UART signal path. I've seen some devices that do indeed want an inverted UART signal set.



■ SCHEMATIC 2. This is the design behind the hardware you see in Photo 3.